ARGA: Approximate Reuse for GPGPU Acceleration

Daniel Peroni, Mohsen Imani, Hamid Nejatollahi†, Nikil Dutt†, Tajana Rosing
University of California San Diego, La Jolla, CA, 92093, USA
†University of California Irvine, Irvine, CA, 92697, USA

ABSTRACT

Many data-driven applications including computer vision, speech recognition, and medical diagnostics show tolerance to error during computation. These applications are often accelerated on GPUs, but high computational costs limit performance and increase energy usage. In this paper, we present ARGA, an approximate computing technique capable of accelerating GPGPU applications. ARGA provides an approximate lookup table to GPGPU cores to avoid recomputing instructions with identical or similar values. We propose *multi-table parallel lookup* which enables computational reuse to significantly speed-up GPGPU computation by checking incoming instructions in parallel. The inputs of each operation are searched for in a lookup table. Matches resulting in an exact or low error are removed from the floating point pipeline and used directly as output. Matches producing highly inaccurate results are computed on exact hardware to minimize application error. We simulate our design by placing ARGA within each core of an Nvidia Kepler Architecture Titan and an AMD Southern Island 7970. We show our design improves performance throughput by up to 2.7× and improves EDP by 5.3× for 6 GPGPU applications while maintaining less than 5% output error. We also show ARGA accelerates inference of a LeNet NN by 2.1× and improves EDP by 3.7× without significantly impacting classification accuracy.

CCS CONCEPTS

• **Computer systems organization** → **Multicore architectures**; • **Computing methodologies** → Machine learning approaches.

KEYWORDS

Approximate computing, GPGPU, Floating point unit, Hardware Acceleration

ACM Reference Format:
Daniel Peroni, Mohsen Imani, Hamid Nejatollahi†, Nikil Dutt†, Tajana Rosing. 2019. ARGA: Approximate Reuse for GPGPU Acceleration. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3316781.3317776

1 INTRODUCTION

In recent years the amount of data produced by devices has risen exponentially. The world produces 2.5 quintillion bytes of data per day and over 90% of all data was produced in the last 2 years [1]. The rate of data production is outpacing our ability to process it efficiently on current hardware [2]. Many modern applications including computer vision, machine learning, web searches, speech recognition, and medical diagnostics are highly parallelizable and can be sped up through the use of GPU computing. Running these applications on GPUs can be power intensive, so novel architectures are needed to optimize performance and reduce energy use.

A notable attribute of the previously mentioned applications is the fact they often do not require perfect accuracy in their results [3–5]. In multimedia applications, the output can have small audio or visual artifacts without being noticeable to a human user. Web search algorithms often need to provide the top best matches, but not necessarily in an exact order. NNs and other machine learning algorithms can tolerate small errors without significantly impacting the overall classification accuracy [6–11]. Approximate computing is a method of exploiting error tolerance to trade application accuracy for energy savings and performance improvements. Within machine learning, neural network based solutions have proven to be state of the art for many of these applications [12, 13]. In heterogeneous systems, GPUs are capable of accelerating NNs orders of magnitude faster than CPUs. Despite the performance improvements, neural networks are growing in depth and require further hardware acceleration [14]. Faster and more energy efficient hardware is needed to train and deploy these networks effectively.

Associative memories or lookup tables can be used to reduce the energy consumption of parallel processors by enabling computational reuse [15]. Associative memories are capable of storing frequent arithmetic operations and reusing them instead of computing the value. Applications involving a high percentage of temporal locality can utilize this memory to eliminate redundant computation. Prior work extended computational reuse to approximate computing by enabling inexact matching in the associative memory [15–17]. Previous approximate approaches using associative memory have several notable flaws. They place the memory within the floating point unit (FPU) pipeline and perform searches sequentially. This links them to the FPU pipeline stages and prevents them from accelerating applications, only providing energy savings. They also only consider individual threads, rather than warps as a whole. GPUs issue instructions in groups of threads called a warp. A single lookup miss in one thread of the warp prevents the entire warp from being accelerated.

In this paper, we present ARGA, an approximate computing technique capable of accelerating GPGPU applications. We place Content Addressable Memory (CAM) blocks in GPGPU cores to exploit temporal locality of the applications and avoid recomputing similar values. Searches from the table resulting in an exact or low error match are removed from the floating point pipeline and used directly as output. Highly inaccurate match results are computed on exact hardware to minimize application error. We propose *multi-table parallel lookup* which enables computational reuse to significantly speed-up GPGPU computation by checking up to 5 incoming instructions in parallel. We simulate our design by placing ARGA within each core of an Nvidia Kepler Architecture Titan and an AMD Southern Island 7970. We show our design improves performance throughput by up to 2.7× and improves EDP by 5.3× for 6 GPGPU applications while maintaining less than 5% output error compared to unmodified GPUs. We also show ARGA accelerates inference of a LeNet NN by 2.1× and improves EDP by 3.7×.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317776

2 RELATED WORK

Hardware-based approximate computing can be implemented in a variety of ways including voltage overscaling, approximate arithmetic units, and lookup based approximate matching. Voltage overscaling (VOS) reduces the voltage for a circuit until timing errors start to appear [15, 18]. The benefits of VOS can be improved by designing circuits with reduced critical paths and better-scaling properties [19, 20]. The exponential rate errors increase as voltage decreases and significant transistor leakage currents limits VOS usage. Alternatively, approximate arithmetic units can be used to improve performance. These include approximate adders, multipliers, and dividers which trade precision for energy savings and performance improvements [3, 5, 21, 22]. Approximate computational units show difficulty controlling error, often requiring expensive post-processing steps to correct.

Lookup-based approximation avoids computation entirely by reusing previously computed values. Ternary content addressable memories (TCAMs) can be utilized for approximate computational reuse for GPGPU applications [15, 16, 23, 24]. Associative memory is placed adjacent to FPUs and stores input and output values from previously computed operations which are searched against incoming inputs. The memory returns the nearest distance match from each search, allowing power savings in applications with many repeated calculations. Developers specify regions of their programs able to tolerate approximation, then profile the code to find the most commonly occurring operations. The profiled values are stored in memory prior to runtime and remain static as the program runs. The authors in [15] proposed a configurable memory which applies VOS to non-volatile associative memory to relax computation and to trade output accuracy for energy savings. Work in [23] designed a novel associative memory based bloom filters. Machine learning algorithms and neural networks have proven to be resilient to some level of reduced precision. Networks can be quantized or use FP16 precision to improve performance [25, 26]. Prior works utilized TCAMs to accelerate the NN computation [27]. The networks are pre-profiled for the most frequently occurring operations prior to runtime. During training, the most common values are searched and the closest matches used instead of computing on hardware. Existing work suffers from low hit rates because the static values cannot adapt to variations in running applications. Previous work in memory based computation has several problems. They are tied to the pipeline stage of the floating point unit which prevents them from accelerating application. In addition, they either use static tables which produce low hit rates and require large sizes. Associative memory can be updated through online training [28], however, the process has overhead penalties and suffers from energy costs.

We attempt to address prior works' shortcomings to improve performance of applications. Our design, ARGA, is a dynamically updating lookup table capable of accelerating GPGPU applications and reducing energy consumption. Unlike prior work, ARGA is capable of searching for multiple incoming operations in parallel per core to improve application performance and reduce energy usage. ARGA provides warp level acceleration by identifying and avoiding situations when a small number of threads create a bottleneck.

3 ARGA DESIGN

3.1 Motivation and Overview

A wide range of GPGPU applications, such as machine learning and multimedia, can tolerate some error in their output. At the same time, the operations in these applications show high temporal locality with many identical or similar values being recomputed. Figure 1 shows a breakdown of arithmetic operations computed for the ImageNet

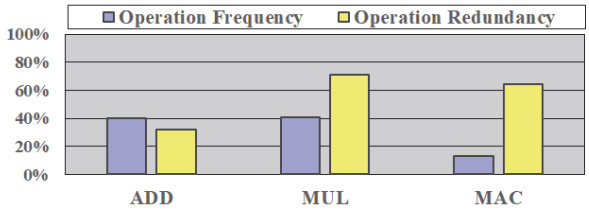


Figure 1: Frequency of operations and percentage of redundant computations in Rodinia [29] backpropagation benchmark

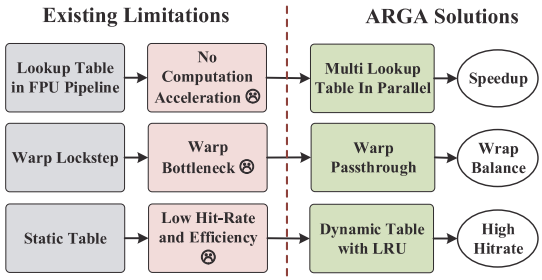


Figure 2: Limitations of the existing approaches compared to ARGA.

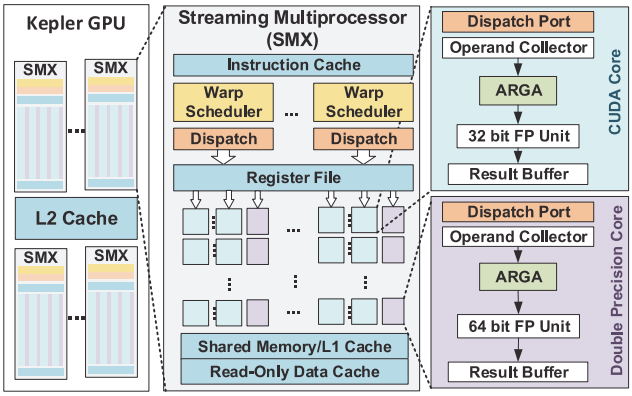


Figure 3: Implementation of ARGA alongside FPU within Nvidia Kepler Titan GPU

backpropagation application from the Rodinia [29] benchmark suite. 95% of the ALU operations involve an add, multiply, or multiply-accumulate, so we focus our efforts on reducing their usage. For each of these operations, we also examine redundancy of the computation. An operation is considered redundant if it is identical or highly similar to a previous calculation. For example, if an FPU first computes 2.0×2.0 followed by 2.02×2.0, the second would be redundant. 70% of the multiplier and 64% of the MAC operations' computed results in the backpropagation application are within 5% of a value from an operation computed in the last 16 instructions. This high level of redundancy can be exploited to improve performance.

Small lookup tables placed adjacent to each FPU avoid recalculating values by storing the most common inputs and outputs for each application [15, 16]. However, the existing approaches have three main issues as listed in Figure 2: (i) these methods only provide energy savings and fail to accelerate applications. (ii) They also fail to account for matching at a warp level. GPU instructions

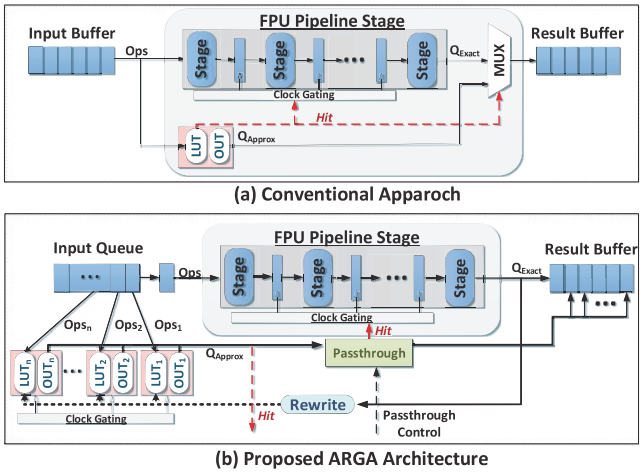


Figure 4: (a) Computational reuse in conventional FPU within GPU (b) ARGA architecture integrated lookup tables with FPU.

within a warp execute in lockstep, which means any acceleration must maintain instruction adherence. (iii) These designs utilize static lookup tables which require applications be pre-profiled to identify the most frequent operations prior to runtime. In addition, static tables result in low lookup table hit rate in practice. In this paper, we propose several novel approaches to address all major practical issues in approximate computational reuse. For the first time, we propose the idea of *Multi-Table Parallel Lookup* in order to accelerate GPGPU application, rather than just saving energy. We propose *Warp Passthrough* to avoid lockstep warps from being bottlenecked by a small neck of threads producing poor approximate results. Finally, we propose a *dynamic table* and a light-weight policy which enable lookup table values to be updated at runtime based on the local data. In the following section, we explain the details of each proposed approach.

3.2 Multi-Table Parallel Lookup

A critical drawback of prior lookup based designs is the lack of application acceleration. Placing a lookup table within the FPU saves power, but is tied to the pipeline [30]. Approximated operations are clock gated to save power, but still, take space in the pipeline. Only one operation is searched for at a time, rather than checking several operations simultaneously. We place multiple tables outside the FPU to check several instructions concurrently. For each search that returns a usable result, the operation is removed from the input buffer avoiding the FPU altogether and accelerating the application.

We integrate ARGA into an Nvidia Titan GPU based on the Kepler GK110 architecture as shown in Figure 3. The GPU has 14 streaming multiprocessors (SMX), each with 192 single precision and 64 double precision floating point units. We place ARGA within each GPU core immediately after the operand collector, ensuring the values are available for lookup. To build the lookup tables for ARGA, we use CAM, proposed in [30]. ARGA provides nearest distance matches using the inputs from each arithmetic operation run on the GPU core. The distance between the match and the input values determines the output error. Better matches have a lower error, while further distances lead to worse approximation. To control application accuracy, the operations are split between using results from ARGA and running on exact FPU based on the match distance. Users set the maximum allowed error for individual matches to trade off between

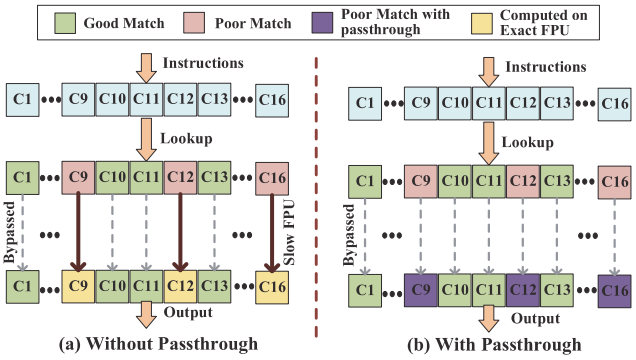


Figure 5: Instructions approximated using a lookup table (a) without passthrough and (b) with the passthrough support.

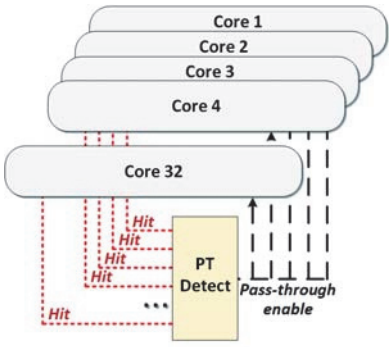


Figure 6: Check of match quality across threads in warp to enable passthrough.

accuracy and performance/energy. Higher error tolerance results in better acceleration and energy savings.

Figure 4(a) shows associative memory used to reduce energy consumption of an FPU [30]. This configuration places the lookup table adjacent to the first stage of the FPU as the search operation takes one cycle. When a match is detected, the FPU stages for the operation are clock gated to save power and the output is used rather than recomputing the value. This approach only checks one operation at a time, preventing application acceleration. ARGA uses multiple lookup tables before the FPU pipeline to search incoming instructions in parallel. Figures 4(b) shows the implementation of ARGA. To accelerate applications, we use *multi-table parallel lookup*, an architecture which compares multiple inputs in lookup tables simultaneously. The lookup table is placed outside the FPU. Incoming operations are stored in a queue before the FPU. Up to 5 of the inputs are searched for in the lookup table at the same time. The state of matches can be one of two states. (i) If one or more searches results in a hit, the stored value associated with the match is output by ARGA. All matched operations are removed from the queue and the remaining ones are moved to the checked queue. (ii) If no matches are detected, values that received the passthrough signal are removed from the queue and the rest are moved to the checked queue. Values in the queue are processed sequentially. As the queue fills, we disable lookup tables to avoid overflows.

The lookup table hit rate has a huge impact on the performance of ARGA. Hit rate is impacted by the level of approximation allowed and the size of the LUT. If three values are processed in parallel and 2 can be approximated, the performance increases by 3×. However,

if the hit rate is low and all three miss, the performance remains the same, but the search operation increases overall energy consumption. The number of lookup tables enabled is adjustable. Each core has multiple tables, but the exact number represents a design trade-off. Additional tables improve acceleration but increase power draw. One method we use to improve our energy savings is to utilize multiple tables, but disable some during periods of low hit rate. As the number of good matches decreases, the checked queue will become full and additional tables are disabled.

3.3 Warp Passthrough

In GPU, instruction threads run in groups called warps. Each core is assigned the same instruction and all operations run in lockstep on a warp. This creates a problem where a single poor match in one of the cores may prevent the entire warp from accelerating that instruction. This is highlighted in Figure 5a, where 3 of the 16 cores bottleneck the operation. Here, individual operations have a higher error than the rest in the warp, but the overall warp error remains low. These operations are computed on exact hardware, increasing energy costs and preventing application acceleration. In order to accelerate the instruction, all operations must be accelerated.

We propose warp passthrough, a scheme which identifies warps with high overall match rates and accelerates them in spite of a few poor matches. Figure 5b shows the same process in which passthrough is enabled. Instead of waiting on a small number of threads to compute on the FPU, all threads in the warp use the result produced by the approximate lookup. This, combined with multi-table parallel lookup, enables ARGA to accelerate applications. As shown in Figure 6 each core in a warp outputs a signal identifying the result produced by the lookup table as either a good or bad match. If a majority of cores identified good matches, it outputs a passthrough enable signal back to the cores. The cores with poor matches send the operation to be computed on exact FPU. The results of the exact operations then update the lookup table using a Least Recently Used (LRU) policy to ensure the most recently computed values are populating the table. Finally, the computed results are reordered and placed in the result buffer.

3.4 Dynamic Updates

Prior work used static tables in order to enable computational reuse [15, 16]. A static table is loaded, prior to runtime, with a set of previously identified common values. To collect these common values, a developer must profile each application and collect the inputs and outputs for each operation. These values are sorted by frequency and the most common are loaded into the associative memory. Using this approach for practical applications results in very low hit rate. For example, images taking during the day have lighter backgrounds, while images taken at night have darker backgrounds. Similarly, the background color may change depending on the season. When processed by a GPU, a single image may have high redundancy, but across two images there may be few similarities. Storing all these possibilities in small static table results in very poor matching.

ARGA is able to update dynamically in order to provide more hits and better matches, leading to better energy savings and improved performance over static lookup tables. ARGA eliminates the pre-profiling step. The values in associative memory are updated regularly using an LRU policy to replace old values. In ARGA, the maximum match distance is adjustable by the user and poor matches exceeding this distance are placed in the FPU queue to be computed exactly. The result calculated by the FPU updates the lookup table. Unlike static tables, dynamic updates can cover previously unseen

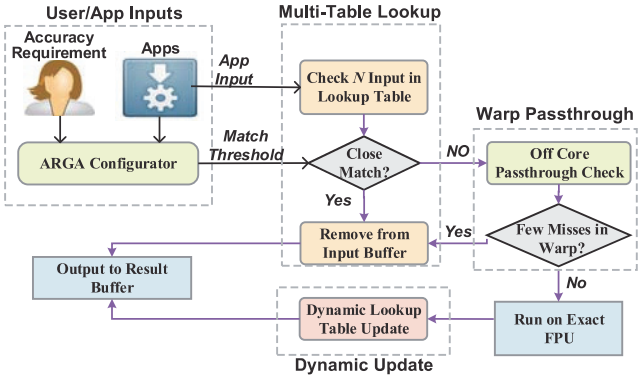


Figure 7: The steps for processing incoming operations.

applications without the need for profiling. One possible disadvantage of the dynamic table is its frequent write operation which can increase the cost of computation. ARGA addresses this issue by enabling long-term write operations which enable table updates with low cost of write energy. In Section 4.2 we explore the impact of the proposed dynamic update on lookup table hit rate.

3.5 ARGA Framework

Figure 7 shows the computation flow of ARGA processing incoming data for an application. The user configures ARGA based on the application and its accuracy requirements. The user sets the maximum match error allowed, with higher values translating to faster performance and energy savings, but worse accuracy. A user adjusts this value to find the optimal configuration per application. At runtime, the application reaches sections of code with arithmetic operations such as multiplier or adder. Lookup tables store the inputs and the outputs computed in exact hardware for recent operations. Incoming inputs are compared to stored values in the table and the nearest distance match is identified. If a match is close, the inputs are removed from the input buffer set and the result stored in ARGA is output to the result buffer. If a close match is not found, the overall miss count of the warp is examined. If a fraction of the cores within a warp do not pass through, a passthrough signal is sent to the passthrough block. In this event, all matches will be sent through the block, even inaccurate ones. If a high fraction of the cores do not provide matches, then these poor matches will be sent to the checked queue and run on the FPU. Finally, The results of these computations are used to update the lookup table contents through LRU replacement.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

To simulate ARGA, we use a modified version of Multi2sim [31], a cycle accurate CPU-GPU simulator. The kernel code is modified to implement the proposed design and enable runtime simulation. To show the generality of our approach, we apply ARGA to two GPU architectures: an Nvidia Kepler GeForce GTX Titan and an AMD Southern Island Radeon HD 7970 device. ARGA is implemented next to the FPU within each of the cores in the GPUs. We implement ARGA for the GPU FPU operations which make up the majority of computation within the tested applications; adder (ADD), multiplier (MUL), and multiply accumulator (MAC). The associative memory used in ARGA is simulated using HSPICE in 45-nm technology with a 1V supply voltage.

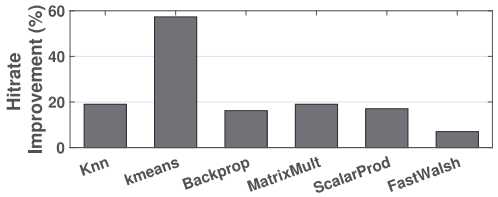


Figure 8: Lookup hitrate improvement using dynamic table.

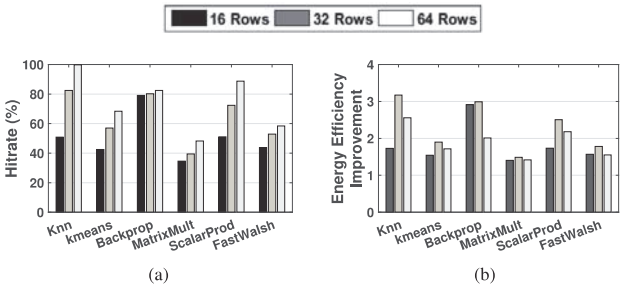


Figure 9: a) The hit rate on ARGA in tested applications and b) the energy improvement for error less than 5%.

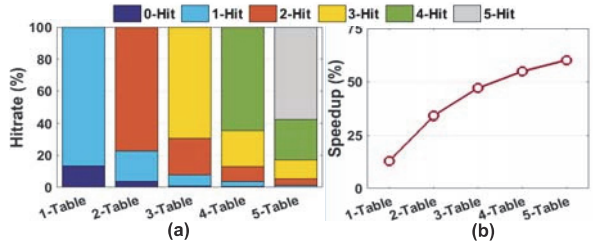


Figure 10: Comparison (a) ratio of hits for increasing the number of tables and (b) the speedup provided for *MatrixMul*

We test the design on Kepler using 3 applications from the m2s-bench-cudasdk-6.5 [31], *MatrixMul*, *FastWalshTransform*, and *ScalarProduct*. For these benchmarks, we use L1/L2 Norm for our accuracy metric. For Southern Island we examine 3 benchmarks from the Rodinia 3.1 machine learning benchmark suite [29], *Back-propagation, K-nearest neighbor*, and *Kmeans*. For these applications, we select average relative error for our accuracy metric. We test the impact of ARGA on a CNN, *LeNet-5* [32], compiled to run on Multi2Sim. The CNN classifies 32x32 pixel images of handwritten digit characters from the `MNIST` dataset [33]. The *LeNet-5* network is trained with 60K training images, and it provides accurate classification for about 97% of 10K tested image samples.

4.2 Dynamic Updating and ARGA Configuration

Figure 8 shows the improvement ARGA provides over a design using a static lookup table[30]. For the tested applications, using a dynamic lookup table provides an average of 22% better hitrate compared to a fixed table using approximate matches of less than 5% error. More matches allow better energy savings and better performance improvements for each application. The number of entries per table (size of a lookup table) impacts ARGA accuracy and efficiency. Increasing the number of entries results in higher hit rates and therefore better performance and energy savings. This comes at the cost of search energy. Larger tables require more energy and time to search and identify the closest match. Figure 9 compares tables ranging from 16 to 64 entries. On average a table size of 32 rows provides better reuse rate than the 16-row table without the excessive power draw of the 64-row table. Further results use ARGA with 32 rows for analysis.

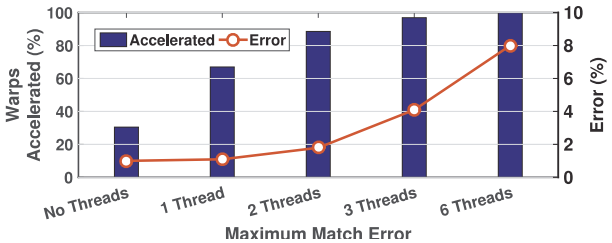


Figure 11: Warps accelerated by ARGA using warp passthrough and the change in output accuracy for *ScalarProd*

4.3 Multi-Table Parallel Lookup

Using more than one table allows ARGA to accelerate applications. We examine the trade-offs associated with increasing the number of lookup tables. More tables increase the area overhead and power consumption of ARGA. Figure 10 shows acceleration as the number of tables increases for the *MatrixMultiply* application with less than 5% error. The application has an overall hit rate of 86%. If 5 tables are used, 49% of checks result in all 5 tables identifying a close match, while the remaining 37% have 4 matches or fewer. Each miss must be computed on exact hardware and prevents speedup. Additional tables provided decreased benefits to performance as the percentage of operations making use of all tables decreases.

4.4 Warp passthrough

We show the acceleration improvement provided by warp pass through. We start without passthrough enabled and show the accuracy and performance improvements. Figure 11 shows the bottleneck of threads for the *ScalarProduct* application. Considered individually, 84% of the operations in the application can be approximated with less than 5% error. However, the warps are bottlenecked, so only 30% can be accelerated. By allowing passthrough for when a single thread produces a poor match, we can accelerate 65% of threads. Although passthrough violates the amount of accuracy which ARGA ensures, it can be used to significantly accelerate the computation.

4.5 Energy Reduction and Acceleration

ARGA provides significant improvements to both acceleration and energy reduction for the tested applications. Figure 12 shows the improvement ARGA provides over an unmodified GPU. As the maximum allowable error per operation is increased, the application finds more matches within ARGA. The performance increases with hit rate, but overall output error also increases. Across the 6 tested applications we show a 2.7× speedup and 5.3× EDP improvement while providing less than 5% application error. Figure 13 shows the output of the *K-means* application. In this example, 91% of the operations are approximate using ARGA, resulting in only 1.1% classification error compared to one run on exact hardware only.

4.6 Convolutional Neural Network

In this section, we show the improvement ARGA can provide on a neural network application. Table 1 shows the speedup and energy savings provided to a NN using ARGA. Compared to the other applications tested, the neural network [32] is much more tolerant to error. The maximum match error can be increased to 25% before significantly impacting output accuracy. We see a rapid drop off in classification accuracy, rather than a linear change as shown by the other applications. For a decrease in classification accuracy of 0.8%,

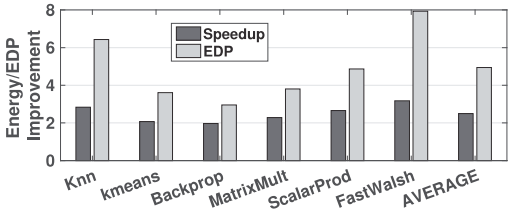
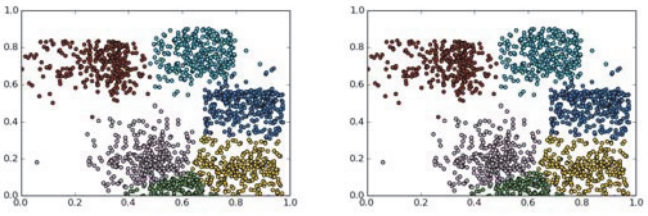


Figure 12: Speedup and EDP improvement provided by ARGA for 6 GPGPU applications.



(a) Exact (b) Approx

Figure 13: Visual comparison of output for *K-means* application running on (a) exact hardware only, (b) ARGA in approximate mode resulting in 1.1% error.

Table 1: Impact of ARGA maximum operation error on the CNN accuracy and efficiency.

Operation Max Error	*Exact*	*5%*	*10%*	*25%*	*50%*	*100%*
Classification Accuracy	97.4%	97.4%	97.0%	96.6%	89.6%	13.8%
Hit rate	23.4%	48.2%	61.9%	75.1%	87.4%	99.8%
Speedup	1.2×	1.5×	1.7×	2.1×	2.5×	2.8×
EDP Improvement	1.2×	1.9×	2.6×	3.7×	5.2×	6.9×

we are able to approximate and accelerate 75.1% of the arithmetic operations. We show a 2.1× speedup and 3.7× EDP improvement.

4.7 Overhead

Here we estimate the overhead of ARGA. The overhead is directly related to the size and the number of lookup tables. Based on our evaluation, ARGA requires five lookup tables with 32-rows in order to provide maximum efficiency. Each lookup table block is a conventional crossbar memory with the nearest search capability. Crossbar memories are transistor-free memories which consist of two memristor devices (0T-2R) [34]. This memory blocks can be integrated at the top of CMOS-based logic with minor area overhead which comes from the sense amplifier. Each floating point multiplier takes $7890\mu m^2$ of area. Therefore, it can fit up to six lookup tables at the top. Since other floating point units take less area, they can fit fewer lookup tables. For example, floating point addition can fit four lookup tables at the top. Considering the overall GPU area, we observe that ARGA using five lookup tables next to each FPGA adds less than 3.8% area overhead to the existing GPU architecture.

5 CONCLUSION

In this paper, we present ARGA, a lookup-based approximate computing technique capable of accelerating GPGPU applications. We avoid redundant computations through the use of associative memory lookups. ARGA exploits the temporal locality of applications in order to avoid computing instructions with identical or similar values. We show *multi-table parallel lookup* speeds up GPGPU computation significantly. We also propose *warp passthrough* and *dynamic lookup table* to avoid lockstep in GPU warps and improve the lookup table hit rates respectively. We show our design improves performance throughput by up to 2.7× and improves EDP by 5.3× for 6 GPGPU applications while maintaining less than 5% output error.

ACKNOWLEDGEMENTS

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1730158 and #1527034.

REFERENCES

[1] C. Dobre and F. Xhafa, "Intelligent services for big data science," *Future Generation Computer Systems*, vol. 37, pp. 267-281, 07 2014.
[2] V. C. Storey and I.-Y. Song, "Big data technologies and management: What conceptual modeling can do," *Data Knowledge Engineering*, vol. 108, pp. 50 – 67, 2017.
[3] J. Han *et al.*, "Approximate computing: An emerging paradigm for energy-efficient design," in *IEEE ETS*, pp. 1–6, IEEE, 2013.
[4] C. Liu *et al.*, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *IEEE/ACM DATE*, p. 95, IEEE, 2014.
[5] S. Hashemi *et al.*, "Drum: A dynamic range unbiased multiplier for approximate applications," in *IEEE/ACM ICCAD*, pp. 418–425, IEEE Press, 2015.
[6] Y. Kim *et al.*, "Efficient human activity recognition using hyperdimensional computing," in *ICIOT*, p. 38, ACM, 2018.
[7] M. Imani *et al.*, "Rapidnn: In-memory deep neural network acceleration framework," *arXiv preprint arXiv:1806.05794*, 2018.
[8] S. Salamat *et al.*, "Rnsnet: In-memory neural network acceleration using residue number system," in *ICRC*, pp. 1–10, IEEE, 2018.
[9] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," in *CLOUD*, pp. 1–6, IEEE, 2019.
[10] Y. Kim *et al.*, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *ICCAD*, pp. 25–32, IEEE, 2017.
[11] S. Salamat *et al.*, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *FPGA*, pp. 53–62, ACM, 2019.
[12] T. C. et al., "A high-throughput neural network accelerator," *IEEE Micro*, vol. 35, pp. 24–32, May 2015.
[13] W. e. a. Liu, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, 12 2016.
[14] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *CoRR*, vol. abs/1707.07012, 2017.
[15] M. Imani *et al.*, "Resistive configurable associative memory for approximate computing," in *DATE*, pp. 1327–1332, IEEE, 2016.
[16] A. Rahimi *et al.*, "Approximate associative memristive memory for energy-efficient gpus," in *DATE*, pp. 1497–1502, IEEE, 2015.
[17] M. Imani *et al.*, "Approximate computing using multiple-access single-charge associative memory," *IETC*, vol. 6, no. 3, pp. 305–316, 2018.
[18] P. K. Krause *et al.*, "Adaptive voltage over-scaling for resilient applications," in *DATE*, pp. 1–6, IEEE, 2011.
[19] A. B. Kahng *et al.*, "Slack redistribution for graceful degradation under voltage overscaling," ASPDAC '10, pp. 825–831, IEEE Press, 2010.
[20] V. K. Chippa *et al.*, "Scalable effort hardware design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 2004–2016, Sept 2014.
[21] M. Imani *et al.*, "Cfpu: Configurable floating point multiplier for energy-efficient computing," in *Design Automation Conference 2017*, p. 76, ACM, 2017.
[22] M. Imani *et al.*, "Rmac: Runtime configurable floating point multiplier for approximate computing," in *ISLPED*, p. 12, ACM, 2018.
[23] X. e. a. Jiao, "Energy-efficient neural networks using approximate computation reuse," in *DATE, 2018*, pp. 1223–1228, IEEE, 2018.
[24] D. Peroni, M. Imani, and T. Rosing, "Alook: adaptive lookup for GPGPU acceleration," in *ASPDAC 2019, Tokyo, Japan*, pp. 739–746, 2019.
[25] I. H. et al., "Quantized neural networks: Training neural networks with low precision weights and activations," *CoRR*, vol. abs/1609.07061, 2016.
[26] P. M. et al., "Mixed precision training," *CoRR*, vol. abs/1710.03740, 2017.
[27] M. S. Razlighi *et al.*, "Looknn: Neural network with no multiplication," in *Proceedings of the Conference on Design, Automation & Test in Europe*, pp. 1779–1784, IEEE/ACM, 2017.
[28] M. Imani *et al.*, "Acam: Approximate computing based on adaptive associative memory with online learning," in *IEEE/ACM ISLPED*, pp. 162–167, 2016.
[29] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009.*, pp. 44–54, Ieee, 2009.
[30] M. Imani *et al.*, "Program acceleration using nearest distance associative search," in *ISQED*, pp. 43–48, IEEE, 2018.
[31] X. Gong, R. Ubal, and D. Kaeli, "Multi2sim kepler: A detailed architectural gpu simulator," in *(ISPASS)*, pp. 269–278, April 2017.
[32] Y. e. a. LeCun, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[33] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2010.
[34] M. Imani *et al.*, "Approximate computing using multiple-access single-charge associative memory," *IEEE TETC*, 2016.